Question 1

a.

The indices of the removed data points (**in the .csv file**) are 21, 43, 111, 146, 232, 303.

```
####QUESTION 1####
## a
data = pd.read_csv('real_estate.csv')
raw_data = data.copy()
lenth = data.shape[0]
index_in_file = []
index_in_code = []
for i in range(lenth):
    if data.iloc[i].isnull().any():
        index_in_file.append(i + 2)
        index_in_code.append(i)
print('The index of removed rows in the file are ',index_in_file)
data = data.drop(index = data.index[index_in_code])
data = data.reset_index(drop = False)
data = data.drop(['transactiondate', 'latitude', 'longitude', 'price'], axis = 1)
```

b.

The mean value of feature ' age ' = 0.40607932670785213

The mean value of feature ' nearestMRT ' = 0.16264267697310722

The mean value of feature ' nConvenience ' = 0.4120098039215686

```
##b
features = ['age', 'nearestMRT', 'nConvenience']
for feature in features:
    minx = min(data[feature])
    maxx = max(data[feature])
    for i in data.index:
        data.loc[i, feature] = (data.loc[i,feature] - minx) / (maxx - minx)
mean_feature = {}
for feature in features:
    mean_feature[feature] = np.mean(data[feature])
    print('The mean value of feature \'', feature, '\' = ', mean_feature[feature])
```

Question 2

The first row of the training set is
 age 0.730594
nearestMRT 0.009513
nConvenience 1.000000

The last row of the training set is
 age 0.878995
nearestMRT 0.099260
nConvenience 0.300000

The first row of the testing set is
 age 0.262557
nearestMRT 0.206780
nConvenience 0.100000

The last row of the tesing set is
 age 0.148402
nearestMRT 0.010375
nConvenience 0.900000

```
lenth = data.shape[0]
cut = lenth // 2
lenth = data.shape[0]
cut = lenth // 2

print('The first row of the training set is \n', data.loc[0][1:],
      '\nThe last row of the training set is \n', data.loc[cut - 1][1:],
      '\nThe first row of the testing set is \n', data.loc[cut][1:],
      '\nThe last row of the tesing set is \n', data.loc[cut+cut-1][1:])

#split train set and test set
X = np.zeros((lenth,1,4))
for i in data.index:
    X[i] = [1,data.loc[i, 'age'], data.loc[i,'nearestMRT'], data.loc[i, 'nConvenience']]

y = np.zeros((lenth,1))
for i in range(len(X)):
    raw_index = data.loc[i, 'index']  #the index in raw_data
    y[i] = raw_data.loc[raw_index, 'price']

X_train = X[0 : cut]
X_test = X[cut : cut + cut]
y_train = y[0 : cut]
y_test = y[cut : cut + cut]
```

Question 3:

$$L_c(y,\hat{y}) = \frac{1}{n}\sum_{i=1}^{n} L_c(y^{(i)},\hat{y}^{(i)}) = \frac{1}{n}\sum_{i=1}^{n}\left[\sqrt{\frac{1}{c^2}\left(y^{(i)} - \left\langle w^{(t)}, X^{\langle i\rangle}\right\rangle\right)^2 + 1} - 1\right]$$

Let $h_w = \left\langle w^{(t)}, X^{\langle i\rangle}\right\rangle$, then we have:

$$\frac{\partial L_c\left(y^{(i)},\hat{y}^{(i)}\right)}{\partial w_k} = \frac{1}{n}\sum_{i=1}^{n}\frac{\partial\sqrt{\frac{\left(y^{(i)}-h_w\right)^2}{c^2}+1}}{\partial w_k} = \frac{1}{n}\sum_{i=1}^{n}\frac{\frac{\partial\left[\frac{\left(y^{(i)}-h_w\right)^2}{c^2}+1\right]}{\partial w_k}}{2\sqrt{\frac{\left(y^{(i)}-h_w\right)^2}{c^2}+1}}$$

$$\frac{\partial\left[\frac{\left(y^{(i)}-h_w\right)^2}{c^2}+1\right]}{\partial w_k} = \frac{\partial\left[\frac{\left(y^{(i)}-h_w\right)^2}{c^2}\right]}{\partial w_k} = \frac{1}{c^2}\frac{\partial\left(y^{(i)}-h_w\right)^2}{\partial w_k} = \frac{2\left(y^{(i)}-h_w\right)}{c^2}\frac{\partial\left(y^{(i)}-h_w\right)}{\partial w_k} = \frac{2\left(h_w-y^{(i)}\right)}{c^2}\frac{\partial h_w}{\partial w_k}$$

Recall that $h_w = \left\langle w^{(t)}, X^{\langle i\rangle}\right\rangle$, so $\frac{\partial h_w}{\partial w_k} = x_k^{(i)}$

Thus

$$\frac{\partial L_c\left(y^{(i)},\hat{y}^{(i)}\right)}{\partial w_k} = \frac{\frac{2\left(h_w-y^{(i)}\right)}{c^2}x_k^{(i)}}{2\sqrt{\frac{\left(y^{(i)}-h_w\right)^2}{c^2}+1}} = \frac{\left(h_w-y^{(i)}\right)x_k^{(i)}}{c^2\sqrt{\frac{\left(y^{(i)}-h_w\right)^2}{c^2}+1}} = \frac{\left(h_w-y^{(i)}\right)x_k^{(i)}}{c\sqrt{\left(y^{(i)}-h_w\right)^2+c^2}}$$

Since $h_w = \left\langle w^{(t)}, X^{\langle i\rangle}\right\rangle$, now we have

$$\frac{\partial L_c\left(y^{(i)},\hat{y}^{(i)}\right)}{\partial w_k} = \frac{\left(\left\langle w^{(t)}, X^{\langle i\rangle}\right\rangle - y^{(i)}\right)x_k^{(i)}}{c\sqrt{\left(y^{(i)}-\left\langle w^{(t)}, X^{\langle i\rangle}\right\rangle\right)^2+c^2}} = \frac{x_k^{(i)}\left(\left\langle w^{(t)}, X^{\langle i\rangle}\right\rangle - y^{(i)}\right)}{c\sqrt{\left(\left\langle w^{(t)}, X^{\langle i\rangle}\right\rangle - y^{(i)}\right)^2+c^2}},$$

with k = 0, 1, 2, 3.

Question 4:

GD:

$$init\_state = w_0$$

$$for\ k\ in\ [1,2,3]:$$

$$w_{k(j)} = w_{k-1(j)} - \eta \frac{\partial}{\partial w_{k-1(j)}} L_c\left(y, \hat{y}\right), \quad for\ every\ j$$

i.e. $$w_{k(j)} = w_{k-1(j)} - \eta \frac{1}{n} \sum_{i=1}^{n} \frac{\partial}{\partial w_{k-1(j)}} L_c\left(y^{(i)}, \hat{y}^{(i)}\right), \quad for\ every\ j$$

SGD:

$$init\_state = w_0$$

$$for\ k\ in\ \left[1,2,3\right], \{$$

$$for\ i = 1\ to\ m, \{$$

$$w_{k(j)} = w_{k-1(j)} - \eta \left(\frac{\partial}{\partial w_{k-1(j)}} L_c\left(y^{(i)}, \hat{y}^{(i)}\right) \right), for\ every\ j$$

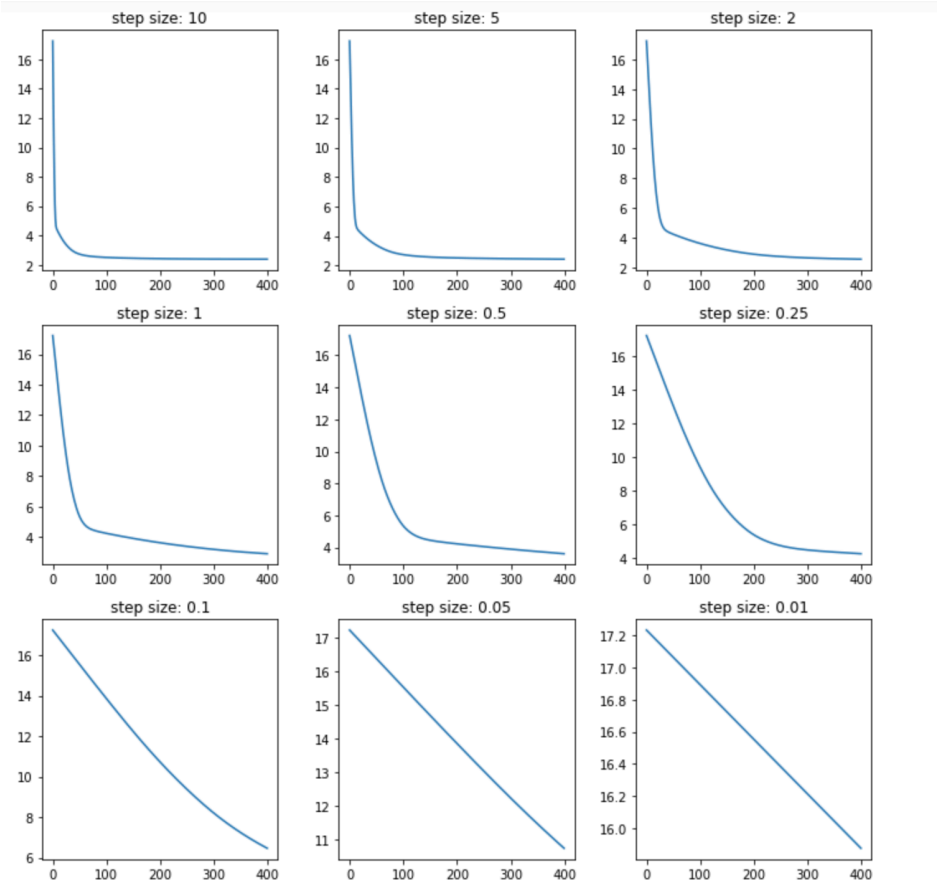
$$\}$$

$$\}$$

Where ‘m’ is the number of observations.

Question5:

a.




```
losses = []

fig, ax = plt.subplots(3, 3, figsize=(10, 10))
nIter = 400
alphas = [10, 5, 2, 1, 0.5, 0.25, 0.1, 0.05, 0.01]

for eta in alphas:
    w0 = np.ones((4, 1))
    loss_per_iter = []
    for iteration in range(nIter):
        loss_all_row = 0
        partial_matrix = np.zeros((4, 1))
        for i in range(len(X_train)):
            dot_item = float(np.dot(X_train[i], w0))
            loss_all_row += (1 / 4 * (float(y[i]) - dot_item) ** 2 + 1) ** 0.5 - 1
            # update w0
            for j in range(len(partial_matrix)):  # for each w in w0
                partial_matrix[j] += X_train[i][0][j] * (dot_item - float(y[i])) / (
                        2 * (((dot_item - float(y[i])) ** 2 + 4)) ** 0.5)
        loss_per_iter.append(loss_all_row / len(X_train))
        w0 = w0 - eta * (1 / len(X_train)) * partial_matrix
    losses.append(loss_per_iter)

for i, ax in enumerate(ax.flat):
    ax.plot(losses[i])
    ax.set_title(f"step size: {alphas[i]}")
plt.tight_layout()
plt.show()

# for i in range(len(losses)):
#     print('step size = ', alphas[i])
#     print(losses[i][-10:])
```

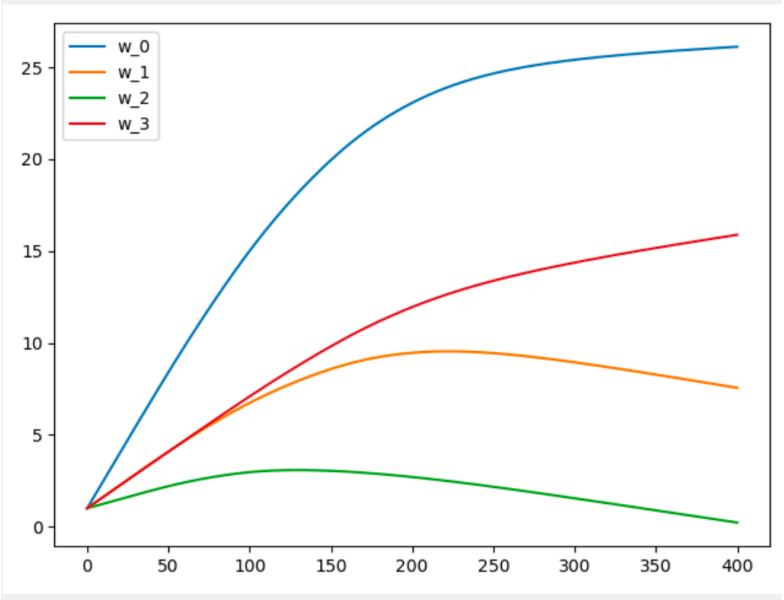
b.

I choose 10 as the step size. By printing the losses of the last 10 iterations for each step size, it could be found that when step size was set to 10, the final loss (after 400 iterations) is the minimum. In addition, with step size = 10, the loss keeps declining gradually within 400 iterations (which is also proved by the curve). This means that even with step size = 10, the global minimum point would not be skipped. Thus I choose step size = 10, so that the algorithm can converge faster, and does not stop the algorithm from convergence. If the number of the iterations grows, it would be reasonable to reduce the step size appropriately, in order to get a better convergence.

```
for i in range(len(losses)):
    print('step size = ', alphas[i])
    print(losses[i][-5:])
```

```
step size =  10
[2.3927018418258172, 2.3926608150333175, 2.392620304610336, 2.3925803040813585, 2.3925408070502443]
step size =  5
[2.4277503662285844, 2.4275214736661677, 2.4272938741798584, 2.427067561002048, 2.4268425274036134]
step size =  2
[2.5593420321587272, 2.558745222802191, 2.5581526861404136, 2.5575643684588902, 2.556980216854798]
step size =  1
[2.9015885745622314, 2.899413285131944, 2.8972514665596516, 2.8951030543998875, 2.892967983619279]
step size =  0.5
[3.62493851175182, 3.6223039821454837, 3.6196742150619516, 3.6170492074728555, 3.6144289561274814]
step size =  0.25
[4.238886433565084, 4.237039240386922, 4.235195268508821, 4.233354456441879, 4.231516744594863]
step size =  0.1
[6.522263690510315, 6.508037441211672, 6.493875749943878, 6.479778369830194, 6.465745048736023]
step size =  0.05
[10.787560137156808, 10.773213188262783, 10.758882181452568, 10.744567112497277, 10.730267978499299]
step size =  0.01
[15.888484301740444, 15.885082909455905, 15.881681563192704, 15.8782802631048, 15.87487900934692]
```

C.

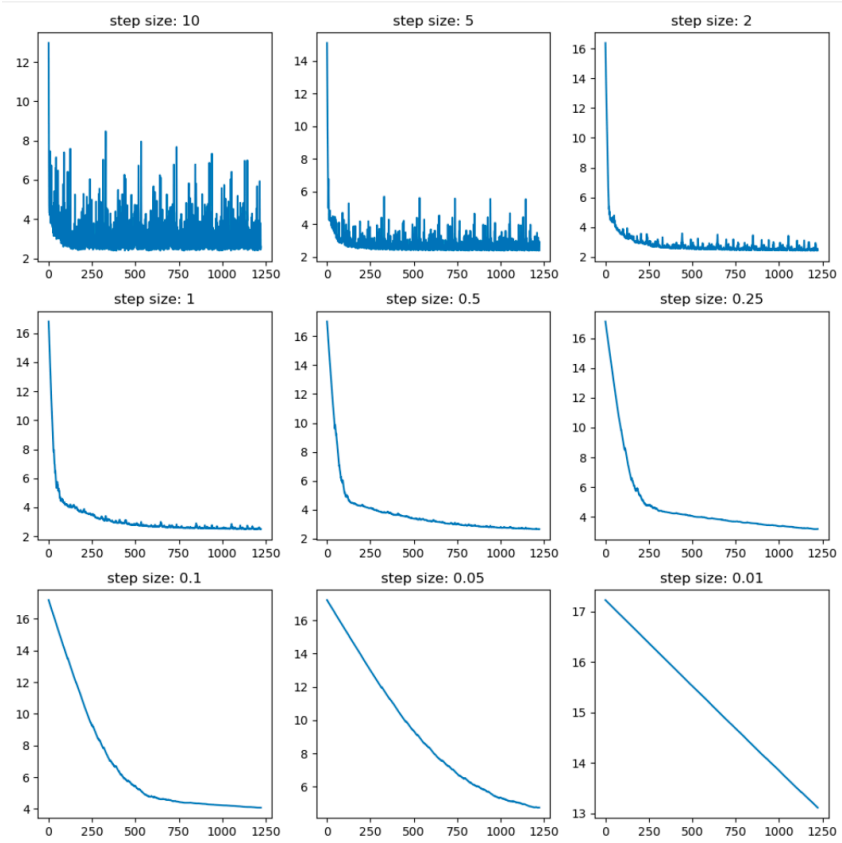



```
The final weight vector is:
 [[26.11838076]
 [ 7.55495429]
 [ 0.22091267]
 [15.88216107]]
The loss on the train set is  4.089850739201337
The loss on the test set is  3.827500929218835
```

```python
eta = 0.3
w0 = np.ones((4,1))
w_list = []
w_list.append(w0)
for iteration in range(nIter):
    loss_all_row = 0
    partial_matrix = np.zeros((4,1))
    for i in range(len(X_train)):
        dot_item = float(np.dot(X_train[i], w0))
        loss_all_row += (1/4 * (float(y_train[i]) - dot_item)**2 + 1)**0.5 - 1
        #update w0
        for j in range(len(partial_matrix)):   #for each w in w0
            partial_matrix[j] += X_train[i][0][j]*(dot_item - float(y_train[i])) / (2*(((dot_item - float(y_train[i]))**2 + 4))**0.5)
    w0 = w0 - eta * (1/len(X_train)) * partial_matrix
    w_list.append(w0)

print('The final weight vector is: ', w_list[-1])
w_list = np.array(w_list)
w_t = w_list.T[0]
for i in range(len(w_t)):
    plt.plot(w_t[i], label = ['w_0', 'w_1', 'w_2', 'w_3'][i])
    plt.legend()
plt.show()

train_loss = 0
w0 = w_list[-1]
for i in range(len(X_train)):
    dot_item = float(np.dot(X_train[i], w0))
    train_loss += (1/4 * (float(y_train[i]) - dot_item)**2 + 1)**0.5 - 1
print('The loss on the train set is ', train_loss / len(X_train))

test_loss = 0
for i in range(len(X_test)):
    dot_item = float(np.dot(X_test[i], w0))
    test_loss += (1/4 * (float(y_test[i]) - dot_item)**2 + 1)**0.5 - 1
print('The loss on the test set is \n', test_loss/len(X_test))
```

Question 6

a.




```
epoch = 6
losses = []

fig, ax = plt.subplots(3, 3, figsize=(10, 10))
nIter = 400
alphas = [10, 5, 2, 1, 0.5, 0.25, 0.1, 0.05, 0.01]

for eta in alphas:
    w0 = np.ones((4, 1))
    loss_per_eta = []
    partial_matrix = np.zeros((4, 1))
    for iteration in range(epoch):
        for i in range(len(X_train)):
            dot_item = float(np.dot(X_train[i], w0))
            # update w0
            for j in range(len(partial_matrix)):  # for each w in w0
                partial_matrix[j] = X_train[i][0][j] * (dot_item - float(y_train[i])) / (
                            2 * (((dot_item - float(y_train[i])) ** 2 + 4)) ** 0.5)
            w0 = w0 - eta * partial_matrix

            loss_all_row = 0
            for k in range(len(X_train)):
                dot_item = float(np.dot(X_train[k], w0))
                loss_all_row += (1 / 4 * (float(y_train[k]) - dot_item) ** 2 + 1) ** 0.5 - 1
            loss_per_eta.append(loss_all_row / len(X_train))
    losses.append(loss_per_eta)

for i, ax in enumerate(ax.flat):
    ax.plot(losses[i])
    ax.set_title(f"step size: {alphas[i]}")  # plot titles
plt.tight_layout()  # plot formatting
plt.show()
```

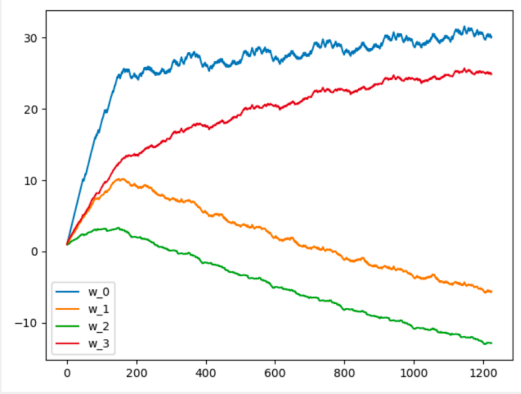
b.

I choose 1 as the step size. According to the curves (in question6.a) of different step size, it could be concluded that if the step size is less than 0.25, the algorithm seems cannot converge within the specific number of iterations. The larger the step size is, the curve tend to be less smooth. This means the global minimum point may be skipped with such step size. With step size = 1, the algorithm seems converge well, the curve is smooth enough, and the loss is small enough. Thus I choose step size = 1.

```
for i in range(9):
    print('step size = ', alphas[i])
    print(losses[i][-10:])
```

```
step size =  10
[2.6272698430458035, 4.986623770856904, 5.9343890709185905, 3.3580952382304954, 2.4773334842545576, 3.13067112209578
9, 2.4477424405488493, 3.7938564298954547, 2.471701524088955, 2.596738520994503]
step size =  5
[2.4195648979450 45, 2.607807802760025, 3.198011342890384, 2.5339591989239083, 2.4084244379539617, 2.5896911835714733,
2.3938252475897595, 2.6835114219439298, 2.9573071088328478, 2.4122552203915952]
step size =  2
[2.5450588858216325, 2.4344450861695064, 2.425864825139093, 2.436295460584729, 2.4355306446139724, 2.493196320829506
7, 2.4302450358046865, 2.4361534484486875, 2.5555522726443116, 2.437132220456736]
step size =  1
[2.5521992608589965, 2.508980865703362, 2.4960674148746773, 2.511824415257782, 2.515168546081596, 2.546428675786477,
2.5150260281611905, 2.496418356202346, 2.5005447244861934, 2.494297787694]
step size =  0.5
[2.657429491912484, 2.662918943546366, 2.665855531236705, 2.6585000017977802, 2.658137716771898, 2.6568585624095227,
2.656472000938984, 2.6606032127754724, 2.672974885237244, 2.663502284434713]
step size =  0.25
[3.173550317841913, 3.1786190223506994, 3.179986758254855, 3.1751942775622237, 3.176703690310678, 3.172073438243539,
3.1735041491240055, 3.177602368321483, 3.185899212603759, 3.183781508215195]
step size =  0.1
[4.077236458151265, 4.078700812696992, 4.078427991081444, 4.077245556226937, 4.07663404510091, 4.075476027020809, 4.0
752356217112595, 4.075842231649199, 4.077382908074877, 4.077287872756305]
step size =  0.05
[4.746941697328147, 4.75253874048138, 4.755046679383046, 4.7500314850394725, 4.754289948409087, 4.749213630367751, 4.
752146698056933, 4.756453529889469, 4.750577761332932, 4.753105274565418]
step size =  0.01
[13.14140605430685, 13.137720013542996, 13.134837639331469, 13.131556173753946, 13.127903157334881, 13.12458054070959
8, 13.121557447983045, 13.118472682640146, 13.11448560485412, 13.110823677778672]
```

c.




```
The final model is
 [[ 30.13206316]
 [ -5.58368145]
 [-12.85975331]
 [ 24.87535719]]
The loss on the train set is  2.780238696600798
The loss on the test set is  2.8448220512160347
```

```python
w0 = np.ones((4,1))
eta = 0.4
w_list = []
w_list.append(w0)
loss_per_eta = []
partial_matrix = np.zeros((4,1))
for iteration in range(epoch):
    for i in range(len(X_train)):
        dot_item = float(np.dot(X_train[i], w0))
        #update w0
        for j in range(len(partial_matrix)):   #for each w in w0
            partial_matrix[j] = X_train[i][0][j]*(dot_item - float(y_train[i])) / (2*(((dot_item - float(y_train[i]))**2 + 4))**0.5)
        w0 = w0 - eta * partial_matrix
        w_list.append(w0)
w_list = np.array(w_list)
w_t = w_list.T[0]
for i in range(len(w_t)):
    plt.plot(w_t[i], label = ['w_0', 'w_1', 'w_2', 'w_3'][i])
    plt.legend()
plt.show()

train_loss = 0
w0 = w_list[-1]
print('The final model is \n', w0)
for i in range(len(X_train)):
    dot_item = float(np.dot(X_train[i], w0))
    train_loss += (1/4 * (float(y_train[i]) - dot_item)**2 + 1)**0.5 - 1
print('The loss on the train set is ', train_loss / len(X_train))

test_loss = 0
for i in range(len(X_test)):
    dot_item = float(np.dot(X_test[i], w0))
    test_loss += (1/4 * (float(y_test[i]) - dot_item)**2 + 1)**0.5 - 1
print('The loss on the test set is ', test_loss/len(X_test))
```

Question 7

The losses of the train set and the test set of SGD (2.78 and 2.84) are significantly smaller than those of GD (4.09 and 3.83). It could be concluded that SGD has better convergence. It should be noticed that there are only 6 iterations of the entire data in total in SGD, compared with 400 iterations in GD. This shows that the speed of convergence in SGD is much faster than that in GD, as well as has lower loss. In another word, SGD performs better and is more efficient than GD, in this specific task.

The step size is very important in both GD and SGD. If the step size is set too small, the algorithm will converge slower, so that requires more iterations. Namely, it is likely that the algorithm could not converge within a specific number of iterations. However, if the step size is too large, the global minimum point of the loss function may be skipped, resulting in that the curve of the loss waving near the global minimum point eventually but could not reach that point. It is essential to choose a good step size that can weigh the speed and loss well.

GD and SGD have different applying ranges respectively. Since GD uses all data to update the model but SGD uses only one data in each iteration, thus SGD is a better choice when the size of the train data is very large. Similarly, if the number of iterations is so limited that GD cannot converge within the given number of iterations(and step size), SGD may perform better since it converges faster. However, compared with GD, SGD uses only one data to update the model in each iteration, thus the presence of error is unavoidable due to the approximation. If the more accurate minimum value of the loss function is needed, GD will be the preference (as long as after enough iterations).

The characteristic of SGD (use only on data to update the model in each iteration) also explains why the GD paths look much smoother than the SGD paths. The model is updated more frequently (updated when seeing a single data rather than after the whole data set), so that the variance tends to be much higher, resulting in remarkable wave.